PJIM PARSONS JOURNAL FOR INFORMATION MAPPING

THE PARSONS INSTITUTE FOR INFORMATION MAPPING
68 Fifth Avenue New York, NY 10011
212 229 6825 piim.newschool.edu

Transforming the Stock Markets into Music using DataScapR

SAMUEL VAN RANSBEECK

KEYWORDS *Composition, modularity, MaxMSP, sonification, stock market, Yahoo Finance*

PROJECT DATE 2015

URL datascapr.wordpress.com

ABSTRACT DataScapR is a toolbox for stock market sonification in an artistic context. Using real-time and historical data, composers and sound artists are able to use stock market data in their practice, whether that is an artistic or scientific one. DataScapR allows the user to use real-time and historical data and map these data in myriad ways to musical parameters. The mapped data can be used to control VST (Virtual Studio Technology) or MIDI (Musical Instrument Digital Interface) instruments. Furthermore, DataScapR allows the creation of traditional notation scores. As such, DataScapR lets the user harness the dynamics of the stock market and create interesting sonic results.

This paper describes the background and motivation behind DataScapR, explains the different modules and operations, and shows some examples. Finally, future directions are discussed.

1 INTRODUCTION

DataScapR comes from a personal interest in algorithmic composition and the use of extra-musical material in general. This interest, combined with a fascination for the stock markets, made me develop *StockWatch,* a software art piece where stock from various indices were read sequentially and sonified. I described this work in a paper published in this journal in 2009.

While *StockWatch* offered some ways to control the musical outcome, it was fairly limited. In order to use stock market data in a more direct way, the user would have to have more control over what data he could use as well as over the mapping methods. I set out the following objectives: I wanted to create a toolbox that had to be easy-to-use and freely available, and would allow a lot of control of the dataflow. Finally, it should be flexible to allow modification and expansion of the toolbox. To facilitate modifications, the toolbox should be thoroughly documented. Hence, I developed DataScapR, a toolbox of MaxMSP patches that allow the user to control musical parameters using stock market values as input values. I chose MaxMSP as it is one of the most widely used programming environments in the musical world. Furthermore, the visual nature of MaxMSP makes it easy for non-programmers to see how the "under-the-hood" process works.

DataScapR takes on the problem of technology in an artistic environment. While technology has helped composers to explore uncharted musical territories, it can become a burden for that exploration if too much emphasis is placed on the technological aspect. The goal of DataScapR is to offer composers an easy-to-use application to sonify the stock market and integrate this process in their system. While it is possible to use the toolbox out of the box, the modular nature of the system and the implementation in MaxMSP make it possible to extend the toolbox and adapt it to one's wishes.

The toolbox consists of three parts: one to sonify real-time data, another to sonify historical data and a third to transform historical data into symbolic scores. The three modules have many common elements but their distinctive purposes made it necessary to separate them in different parts.

In the following sections, I will describe the background of DataScapR and each of the three parts. At the end I look what the future holds for DataScapR.

1.1 BACKGROUND: WHY THE STOCK MARKETS?

One can ask why I chose to work with stock market data when I was developing DataScapR. This question can be answered on both a subjective-personal and a more objective level.

On a personal-subjective level, stock markets have always attracted me; the idea of becoming rich by trading the stocks at the right moment was, of course, a big reason. Furthermore, the abstraction of the real world into a financial maze seemed interesting; indulging in the stock market tables in the newspapers, one could lose himself and become immersed in a virtual world reigned by data (at that moment nobody was speaking of Big Data yet). Films like *Wall Street* (1987) showed the dynamics of the stock market. Although there were negative sides, the bustling dynamic found in the screaming of traders was compelling and made me want to know more about it. I never pursued the job of a stock trader as I had other interests, but I was fascinated by the whole phenomenon of the stock market.

On the objective level, stock markets can show complex behavior, which can be interesting for musical applications. Much research has been done to determine whether the stock market shows temporal patterns or simply random walks that would make it possible to predict the markets such as Goetzmann (2001) and Agrawal and Tandon (1994). Furthermore, technical analysis of charts is an important methodology to try to predict future movements, and is an important way of analyzing the stock markets. This type of analysis assumes that all information about the stock price is present in the charts themselves. It is beyond the scope of this paper to explain the intricacies of technical analysis, however, a good source on this topic is stockcharts.com.

The financial markets have seen enormous shifts in the last decades. Since the Second World War and certainly with the end of the cold war, globalization has fundamentally changed financial trading. Deregulation of the markets as mandated by Bill Clinton's administration in the 1990s made it possible to shift more capital than ever before. The dotcom bubble burst in the early 2000s and the housing crisis in 2008 have proved that this holds risks.

Fueled by IT-innovation, humans have steadily been replaced by algorithms to perform trades. As such, high-frequency trading, in terms of milliseconds, allows the best algorithm to win instead of the best human trader. Nevertheless, these systems can create havoc like they did in the 2010 flash crisis.[1] High-frequency trading also gives rise to ethical questions. It gives more power to bigger corporations that are able to invest in fast computer systems and in the small timeframes can outplay smaller traders. [2]

Given all the above, we can see that the stock markets are interesting phenomena and can be an interesting venture to explore in an artistic context.

1.2 WHY SONIFY?

Stock market sonification is not new. Worrall gives an excellent account of stock market sonification going back as far as the beginning of the 20th century where traders interpreted the telegraphed signals from the ticker-tape to discern stock symbols and prices. The experiments that Worrall presents are all meant to be used to improve trading and he does not discuss artistic stock market sonifications. However, such work is suggestive of artistic possibilities.

1.2.1 WHAT DATASCAPR IS AND WHAT IT IS NOT

In the process of developing DataScapR, I decided to omit certain features. As developing a toolbox is not a linear process, I stepped back and forth many times to add and remove features. While the absence of features might be an uninteresting element at first, I think explaining those decisions will help in understanding DataScapR's premise better. This motivation of omittance can be seen as well in the perspective of the process of coming-into-being. The artifact (the toolbox) is not the only important element; the development process is a fundamental part of the work.

Not a one-button-application: DataScapR is explicitly not a one- button-application. While it is easy-to-use, the user will have to invest time in setting up all parameters. My goal is to keep the user conscious of his operations on the data while making the software accessible. This is meant to make the user more aware of his actions and their effects on the resulting artifact. In using DataScapR we have to be aware that the compositional artifact is just one aspect of the work; the process of coming-into-being is equally important as the result of the productive activity. If we would only take the result in account then we would "Fetishize the musical work, converting it from a catalyst for experience into a commodity to be traded within an economy, whether financial or ideological."[3] Making the process of DataScapR explicit, every operation in the chain has to be initiated explicitly by the user. While it would be technically possible to create a score by clicking just one button, I believe that the constant engagement with the dataflow will diminish the distance between the composer and the technology and benefit him in his voyage to the final artwork . We can link this idea to Heidegger's idea of breakdowns: only in the breakdown, we become aware of the object's function. The technology becomes human or as John Maeda notes, "Technology needs to be humanised rather than optimised, and yet properly understood on its own terms."[5]

Not a one-stop-shop: DataScapR creates musical material, it does not create a finished artwork. As such, the user has to further sculpt the material in the VST's or in external applications. Related to the previous argument, not delivering a fixed artifact makes the user more aware of his choices and forces him to think about his actions.

No Save function: The most striking omittance is probably DataScapR's lack of a save function.

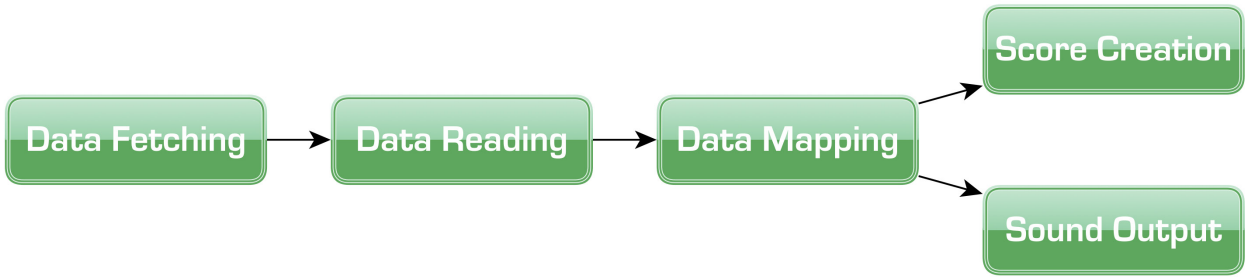

FIGURE 1: *DataScapR: the 5 modules allow the user to follow a logical path from data fetching until the sounding artefact.*

The motivation for this stems from Giorgio Sancristoforo, the creator of Gleetchlab. His statement conveys perfectly why I omitted a save function:

> *I intentionally avoided including save and load functions of gleetchlab settings. (That is since the first version of gleetchlab) Why? It is an important part of my musical approach. In my analog synthesizer days there were no save functions at all but pencil and paper. If you approach each time a reset machine, you are forced to do something new and with little time and patience, you can master the software much better.*[6]

Additionally, the data are always changing. It would thus be illogical to keep a fixed frame for each different dataset we use. There is one instance where I decided to include a save function: in the historical data components, where the user can draw curves, he can subsequently save them and load them later. While the other parameters can be set quite easily, it is impossible to recreate the exact curve used in earlier experiments. I believe that it benefits the user if he can reuse the curves he drew earlier. The addition of the save function is not incompatible with the no save idea: the user will still have to be conscious in setting up the domain and range of the function object. This idea based on Heidegger's breakdowns is thus preserved.

No audio-generation: While I originally intended to include some small patches to generate audio, I removed them later on. There are many good VST's available and it would be a waste of resources to develop something for which there are better versions available.

Besides the fact that there are good VST's available, the addition of audio generation modules could impose a certain style on the composer. For example: if I had included a granular synthesizer, there would be a risk that the composer would feel restrained to use only that type of audio generation. Excluding audio generation is probably the best way to avoid stylistic constraints.

No visualization: DataScapR is intended for sonification. Adding a visualization component would certainly drive attention away from the sonification part. There are many visualization applications and languages available such as *R* and *DataViz*.

2 STRUCTURE OF DATASCAPR'S HISTORICAL DATA PARTS

The historical data patches consist of five modules: data fetching, data reading, mapping, VST or MIDI output and score creation (FIGURE 1). The patches are laid out next to another in a tabbed interface, allowing the user to focus on a specific part of the process.

The DataScapR toolbox allows historical data to be downloaded and used to create compositional material. Yahoo Finance offers historical data on most stocks worldwide and going back as far as 1970.

The datasets can be downloaded directly from the Yahoo Finance site or using the datafetching-patch (FIGURE 2). To get a dataset, the user can select a stock using the menus: Seventeen market indices of markets and their respective stocks and symbols are included in the toolbox by default. Should the user want to track other stocks, he can look up the stock and symbol on the Yahoo Finance page. As stocks are listed and delisted, it is impossible to create a definitive its of all stock symbols. Furthermore, Yahoo formats the stock symbol by adding a suffix for non-American markets. This suffix is not used by all other websites hence it can cause confusion. Therefore, it is advisable to double-check the symbol on the Yahoo webpage.

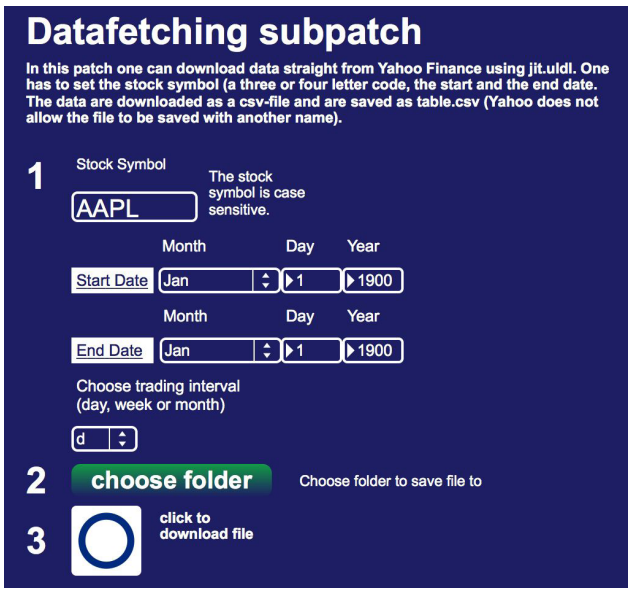

FIGURE 2: *DataScapR: datafetching subpatch window showing three easy steps allow downloading a dataset. The stock symbol can be set manually or by using the umenu's.*

To download the dataset, the user enters the stock symbol, start and end date and the dataset will be downloaded as a csv-file and saved in the chosen folder.[7] The filename is always table.csv and cannot be altered. Downloading a new dataset will overwrite the existing file.

The variables included in the datasets depend on the time resolution (day, week, month) the user chooses.

Daily prices include: the open, high, low, close, and volume for each trading day shown.

Weekly prices include:

- Opening trade of the first trading day for the week.
- High and low price quotes of the week.
- Closing price on the last trading day of the week.
- Adjusted close, based on closing price.
- Weekly volume is the average daily volume for all trading days in the reported week.

Monthly prices include:

- Opening trade from the first trading day of the month,
- High and low price quotes for the month,
- Closing price on the last trading day of the month.
- Adjusted close, based on closing price.

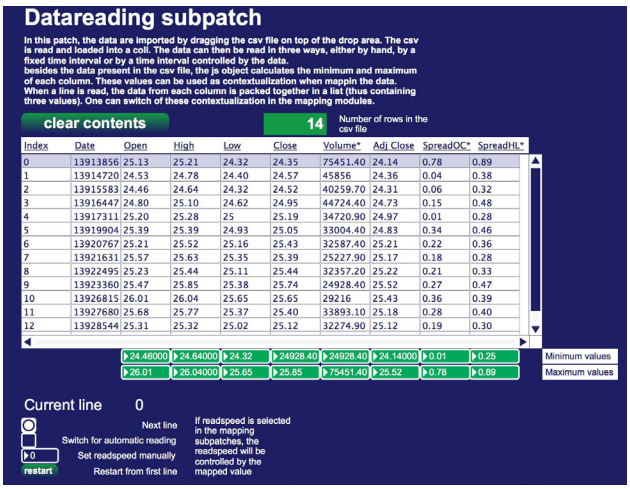

FIGURE 3: *DataScapR: the datareading module allows a visual inspection of the dataset. Minima and maxima are displayed below each column.*

Monthly volume is the average daily volume for all trading days in the reported month. After the dataset is downloaded, the next step is to inspect the dataset. By dropping the csv-file on the jit.cellblock object, the dataset is imported in a coll object and can be inspected visually (FIGURE 3).[8] The csv-file is formatted in reverse-chronological order and comes with a header line. The csv2coll.js javascript object strips the csv file of its header and sorts the data chronologically. The seven parameters are converted from symbols to floats and in addition to those seven parameters, the spreads open-close and high-low are calculated. Finally, the length of the file (the number of data points) is output as well. The date is converted from a symbol in a floating-point number; however, it is not used further down the patch.

After the dataset has been imported and sent to the coll-object, we are ready to read it. This can be done one row at a time or in a dump operation where the whole dataset is sent out in its totality. Here the mapping modules diverge, as they have to handle the incoming values differently.

2.1 MAPPING THE DATA IN THE ROW-BY-ROW COMPONENT

To actually use the data in compositions, we need to map them using the mapping module (FIGURE 4). The mapping module routes the data variables to their output destination.

The data to be mapped are chosen at the left side with a Umenu object (default is the opening price).

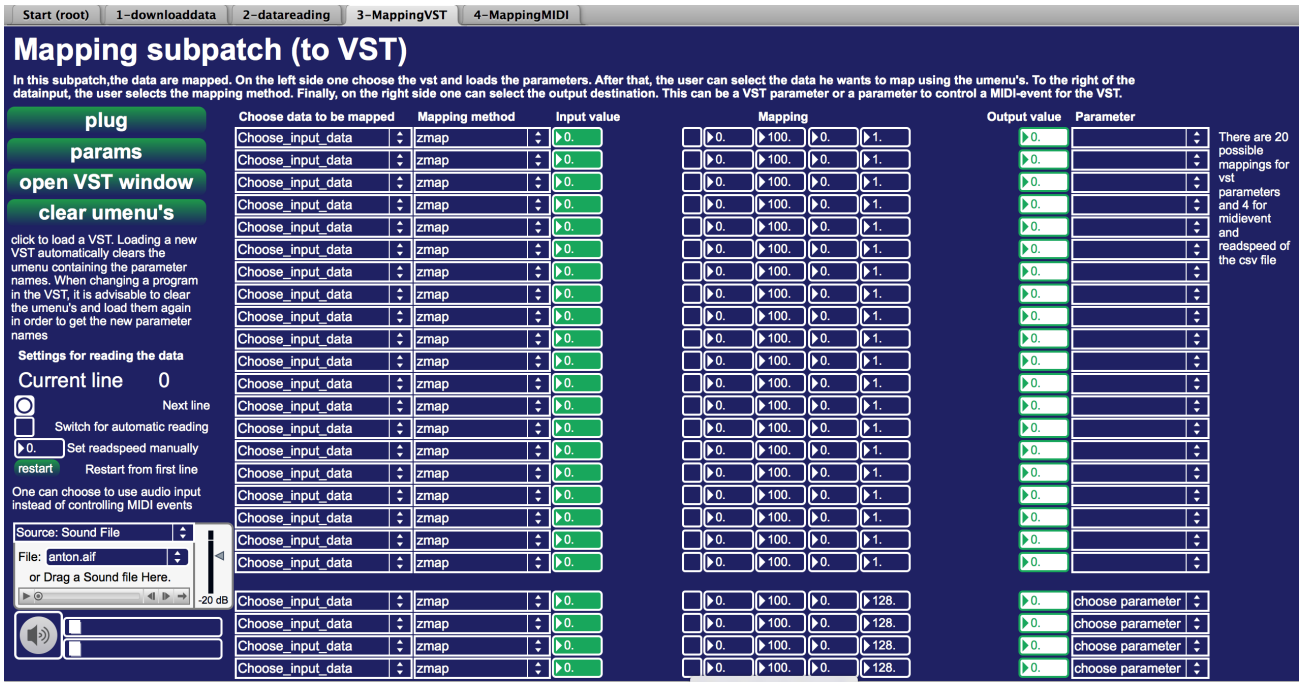

FIGURE 4: *DataScapR: the VST mapping module window*

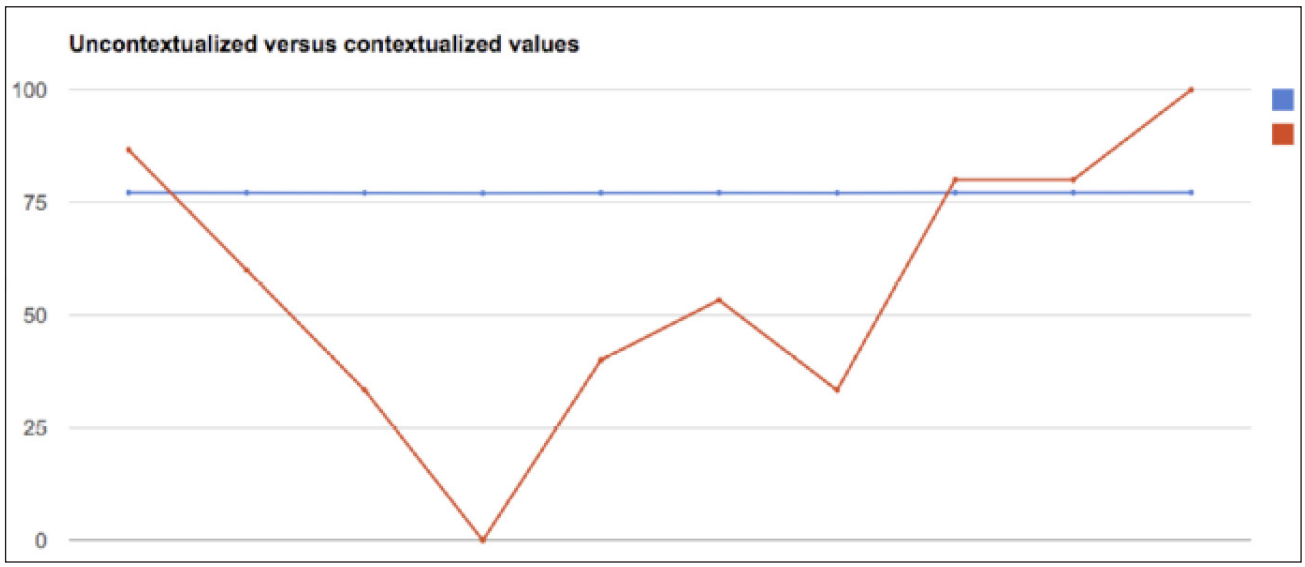

FIGURE 5: *Absolute and contextualized values example: the contextualization makes the profile of the line sharper.*

Uncontextualized	Contextualized values
77.14	86.663956
77.1	59.996948
77.06	33.33
77.01	0
77.07	39.997967
77.09	53.328926
77.06	33.33
77.13	79.995934
77.13	79.995934
77.16	100

TABLE 1: *Absolute and contextualized values example*

After choosing the input data, the mapping method is selected. The first object to the right of the Umenu is a text object that displays the value to be mapped. To the right side of that object are the values and contextualization switch. Finally, at the right side, there is another text object, which displays the first element(s) of the mapped list. In the VST and MIDI patches, one can then choose the output destination through a Umenu.

Four mapping methods are available: a scaling of the incoming values using the Zmap object, a scaling of incoming values using the ej.function object, a modulo operation and a modulo operation followed by a function object.

In the Zmap and function scaling, the incoming values can be contextualized, meaning that the minimum and maximum value in the column is sent to the input range of the Zmap object. The user sets the output range. As such, what could be a small movement in absolute values could be a far bigger movement in relative values. For example, if a stock price oscillates between €77.01 and €77.16, in absolute values this would give a maximum movement of €0.15. If we map the value directly to frequency, the sound result would be relatively static. However, if we extrapolate this €0.15 range to a MIDI range between 0 and 100, the sounding result would be far more dynamic (TABLE 1 and FIGURE 5). The user can use all variables except for the date variable.

Instead of setting the range with a minimum and maximum value, the user can opt to use the function object to create a non-linear mapping. The function object will open in a separate window (FIGURE 6) to avoid cluttering the main patch. Pressing the option key while dragging the segments allows the user to create curves.

A third mapping uses the modulo object, followed by the Zmap object. As Yahoo delivers its data with two decimals, the incoming value is first multiplied by 100 after which the modulo operation is performed. The modulo value can be set by the user.[9] After the modulo operation, the value is sent to a Zmap object as the value needs to be prepared for its output destination. For example, one can perform a modulo 12 operation but then send it to the Zmap object to have the value between 0 and 12 mapped to MIDI-values between 60 and 84. In the case of using modulo, the subsequent Zmap does not use the contextualization from the data but the minimum (0) and maximum value of the modulo operation. Of course, the user can set a custom output range.

The fourth method is essentially the same as the third one, with the difference that instead of a simple Zmap, the function object is used, allowing a more diverse mapping.

All mapping results can be routed to a VSTi (using the VST~object) or a generic MIDI-output. A mapped value can also be sent to the metro object that controls the reading speed of the coll.

2.2 THE VST AND MIDI MODULES

The VST mapping module (FIGURE 4) is quite straightforward; the user loads a VST and then loads all parameters. Loading a new VST will automatically clear the parameter names. When selecting a parameter in one of the menus, that parameter will be automatically disabled in the other menus, which prevents sending out conflicting values. By clicking *open VST window,* the user can open the VST window to inspect the VST visually. In the audio abstraction box, one can choose the audio input to use, as the user wishes. As such, one can use the data to manipulate an audio signal. Clicking the loudspeaker icon at the left side enables the audio; moving the sliders sets the volume.

Finally, one can control the reading of the dataset in three ways:

1. Manually: By clicking the bang button will send out the data on the next line.
2. The dataset can be read at a fixed speed by toggling the switch for automatic reading. Additionally one can set the rate at which a new line is read by using the numberbox.
3. The readspeed can be controlled by a mapped value.

The MIDI patch is essentially the same, only differing in the default output range (between 0 and 128 instead of between 0 and 1).[10] Furthermore, the parameter names are loaded as they are the same for every MIDI instrument.

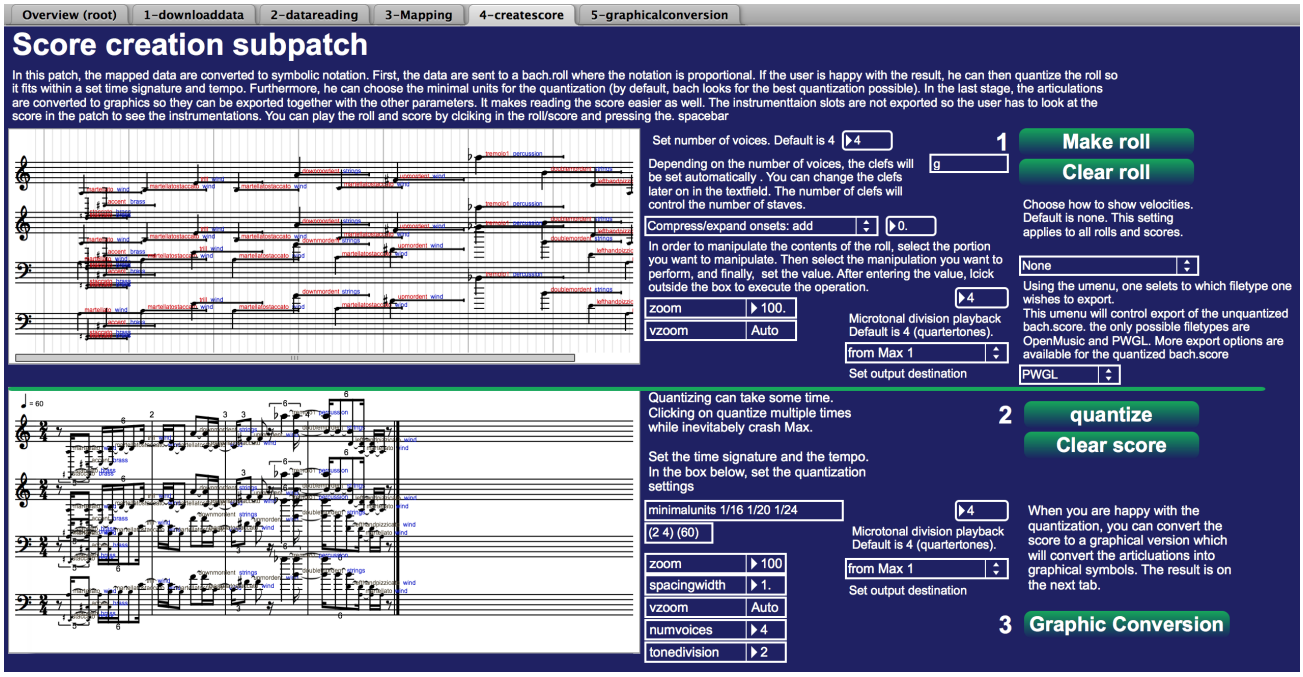

FIGURE 7: *Score creation subpatch window showing the roll and quantized score.*

2.3 SCORE-CREATION PART

Besides reading a dataset row by row, one can create musical scores. The score creation module uses the *bach: automated composer's helper* framework developed by Andrea Agostini and Daniele Ghisi. *Bach* is a set of patches and externals to help in computer-assisted composition. These patches and externals allow the user to quickly create a provisional score so that he can experiment easily with different mappings without having to export the data to another application.

For the score creation, one can control up to four monophonic voices. Every voice has the following parameters that can be controlled by data: pitch, onsets, duration, velocity, articulation, and instrumentation. Not all parameters have to be used; one can disable the mapping by toggling the toggle boxes at the right side of the patch. Instead of sending out the mapped data, the mapping module will send out an empty list, which will result in showing the default value. The default values are shown in the table below.

2.3.1 COMPONENTS OF THE SCORE-CREATING MODULE

The score creation is divided into four patches: data-fetching, data-reading, data-mapping, and score creation. The data-fetching and the data-reading modules are quite similar to the row-by-row module, a key

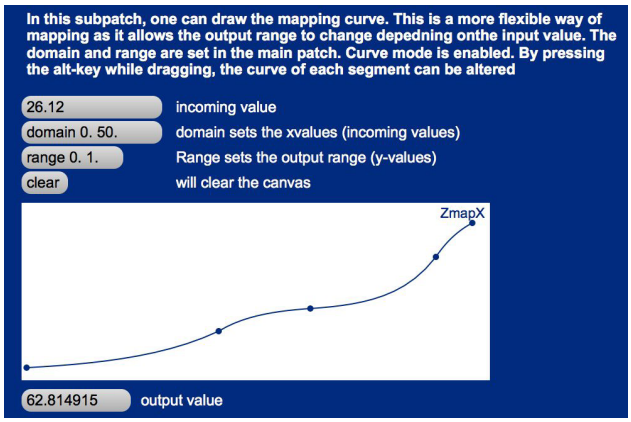

FIGURE 6: *DataScapR: The ej.function object allows non-linear scalings*

difference being that a whole data set is read at high-speed (using an uzi-object) instead of row by row. The mapping methods are the same, however, and the mapping destinations are fixed. When the parameters are not fed with incoming data, bach will output default values. As such, to use only the rhythmical output, one does not feed the pitch parameter and all notes will be central C.

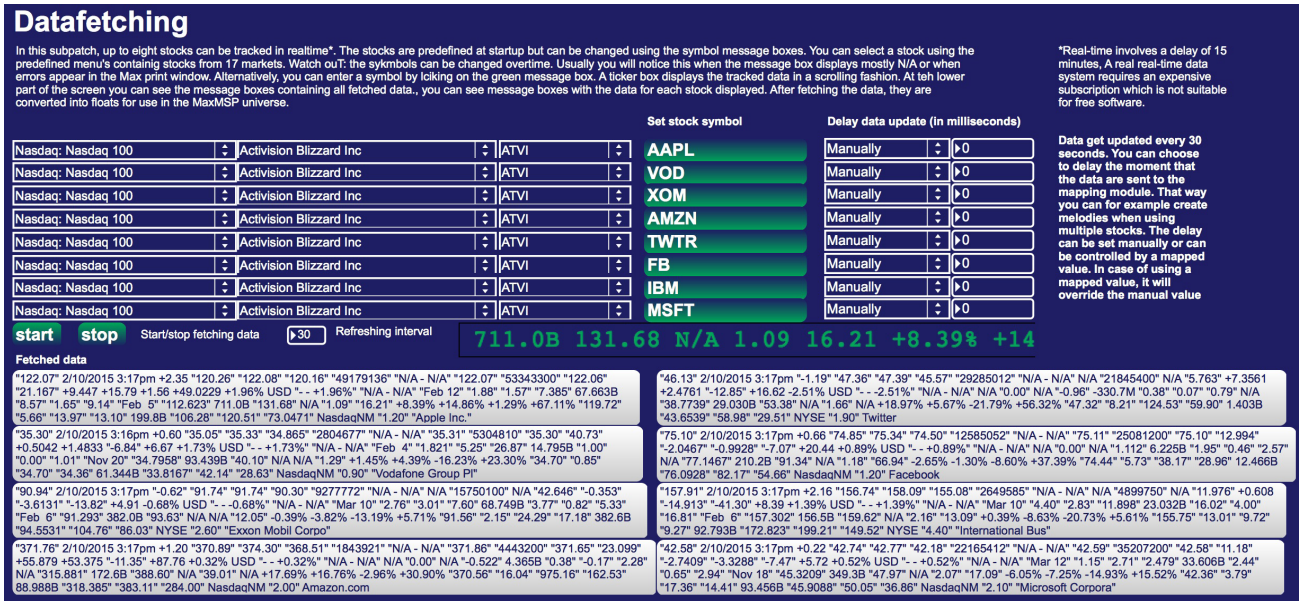

FIGURE 8: *DataScapR real-time datafetching window allowing easy stock selection and a quick visual inspection of the received data*

2.3.2 WORKING WITH THE SCORE CREATION COMPONENT

After importing a dataset, the user can set all the desired mapping parameters in the mapping subpatch. Clicking *map* will trigger an *uzi* object, which will send a series of bangs (the number of bangs being equal to the length of the csv file) to the reading module. As such, the data are mapped sequentially and collected at the end of the mapping in a list. This list is wrapped in parentheses so the Bach framework can use it.[11] As there are four voices, the lists have to be combined using *join object*. Certainly when using large lists, the list might not come in order which could confuse the join object and cause the program to crash. To avoid this, mapping only sets the lists in message boxes through the right inlet. After mapping, the user has to click *combine lists* to trigger the join object.

After having mapped the values and combined the lists, the user can move on to the score creation subpatch (FIGURE 7). Here, the user first sets the number of voices (by default that is four). Clicking the *make roll* button will feed the mapped data into a *bach.roll*, an object that displays the notation in a proportional fashion. If desired, the user can make adjustments by selecting a portion of the roll and move the notes around manually. Alternatively, one can use the menu's to transpose the parameter values by adding, subtracting or multiplying them. Furthermore, one can set the tone division to allow quartertones or even smaller divisions.[12]

In order to obtain a classical score the proportional roll has to be quantized. This is done by clicking the *quantize* button. By default *bach.quantize* is provided with quantize definitions that the developers think are best for general operation, however, the user can change the quantization parameters if desired. (For detailed information how the quantization works, refer to the *bach* help files.) One can set the time signature and the tempo (default is 2/4 and q = 60) Due to *bach*'s handling of articulations the score has to be sent out to another score object to convert the articulations into graphical symbols. The graphical conversion is placed on a separate tab to make the score object bigger and allow easier inspection of the score.

2.3.3 EXPORTING THE SCORE

It is possible to export the unquantized *bach.roll* as a PWGL file or OpenMusic file. The *bach.score* can be exported to more file formats: XML, MIDI, PWGL, OM and Lilypond. The articulations will be exported however the instrumentation will only be visible in the Max patch.

3 REAL-TIME

The real-time patch is divided in four modules: an overview module, data fetching/interpretation and reading, the mapping to VST and MIDI module, and a mixer module.

3.1 DATA-FETCHING MODULE

The data-fetching module is built around a modified java-object (*mxj StockWatch*) from Max's included externals to fetch the data.[13] The *mxj* DataScapR object receives stock market data in almost real-time from the Yahoo Finance API through a dynamic URL.[14] This URL consists of four parts:

1. The basic URL: http://download.finance.yahoo.com/d/quotes.csv?s=
2. The stock symbols are added at the end of the basic URL: http://download.finance.yahoo.com/d/quotes.csv?s=AAPL,GOOG
   Multiple stocks are separated by commas.
3. Each stock property is represented by a letter. Adding these letters will get the desired information. For example, In this URL, the name *n*, stock symbol *s*, Last Trade Price *l1*, the open *o* and closing price *p* are requested using f=nsl1op http://download.finance.yahoo.com/d/quotes.csv?s=GOOG,AAPL&f=nsl1op
4. Finally to download the data, one adds &e=.csv http://download.finance.yahoo.com/d/quotes.csv?s=GOOG,AAPL&f=nsl1op&e=.cs v ("csvQuotesDownload," n.d.)

The mxj object sends an update request data every 30 seconds. Although a higher refreshing rate is possible, this rate seemed the best choice, as it does not overload the server, which would result in service denials. This rate allows sufficient differences in the data. It would not make sense to have to use the same trading price every time when updating every second. Nevertheless, if the user wants to request data at a higher refreshing rate, he can set the desired time interval using the number box located above the ticker.

3.1.1 STOCK PROPERTIES

Yahoo tracks up to 51 properties of a stock such as the last trade price, volume, percent change et cetera (the whole list can be accessed via the menu in the mapping patch). Sometimes less data are available for smaller stocks or more exotic markets. In general we can say that for the main markets, all properties are available. In testing the software, I excluded properties that yielded no data or only showed N/A.

3.2 INTERACTION WITH THE DATAFETCHING

Upon opening the patch (FIGURE 8), the tracking starts automatically with predefined stocks. The user can track up to eight stocks at the same time. To track a specific stock, he clicks on the stock symbol at the upper side of the patch window. A pop-up window will appear, prompting the user to enter the stock symbol. Alternatively, he can choose a stock from 17 market indices worldwide. The data are displayed in a moving ticker as well as message boxes on the lower side of the patch. The output of the mxj.datascapr object is sent through a route object, sending the stock data to the correct message box. From these message boxes, the data are then send to a subpatch where the variables are unpacked, transformed from symbols to floats and sent to the mapping module.

Initially, I included coll objects to store the real-time data. However, one would end up with partial datasets or very small ones unless one lets the system run continuously for a long time. As such, I disconnected the colls from the incoming data. However, if the user wants to record the data, the functionality is present and he would only have to connect the message boxes containing the data to the coll objects.

When the data are fetched they are sent to message boxes from where they are sent to the reading and mapping components. The data for each stock can be delayed so that updates do not go to the mapping modules all at the same time. This can, for example, be useful to create compound melodies.

The data received through the DataScapR object are output as symbols in one string. Hence, they need to be unpacked and converted to floats. This happens in the datastorage and unpacking subpatch (not shown in presentation mode). After the symbol is converted to a value, it is sent out via a send object, for example: the first stock's Last trade is send 1LastTradepriceOnly. In case a data point is not available (N/A), the last received value will be sent out.

3.3 DATA MAPPING

To actually use the data in compositions, we need to map them using the mapping module. This mapping happens in the DatatoVST and DatatoMIDI patches. Each stock is linked to a VST and a MIDI patch. During development, loading the 16 abstractions at start-up resulted in loading times of over a minute. As most users

will likely use one voice to start with, the user can load the abstractions when he needs them. It takes approximately four seconds to load one abstraction, which I deem acceptable. Upon loading, the abstraction is added to the tabbed view at the top of the patcher window.

3.3.1 MAPPING METHODS

The mapping module routes the data to the specific mapping method and subsequently their output destination. The mapping methods are the same as in the historical data parts of the toolbox, explained above. The difference lies in the incoming data: The user can choose 53 + 2 data points to map. While in the historic data components, one can contextualize all data; the nature of the incoming data in the real-time does not allow this. However, Yahoo offers the minimum and maximum day- and year-price. Hence, it is possible to use contextualization for the last trade and I included these options in the data-selection menu.

3.4 OUTPUT

The mapped values can be sent out to a VST or a generic MIDI instrument. This is equal to the row-by-row component. A basic mixer module allows the user to control the audio output volume of each channel.

4 EVALUATION AND FUTURE WORK

As part of the development of DataScapR, I created some works with it: *Vapourwave*s and *4D Brokers*, two installation works, a fixed stereo work, and a piece for solo recorder. These works are documented on the datascapr.wordpress.com blog. In all of these works, I used the toolbox in a distinct way, adapting it for the specific situation.

As shown above, DataScapR offers many possibilities to use stock market data in the creation of artworks. DataScapR is a freely available, easy-to-use and easy expandable toolbox and can be used in a variety of settings. There are still things that can be improved, for example offering more diverse mapping possibilities. But at the moment I think that one can go a long way with DataScapR.

Currently, I am adapting DataScapR to be used with urban data like energy consumption and transit data as part of the MK:Smart project at the Open University in Milton Keynes. The modular nature allows me to adapt the toolbox easily for these new types of datasets. I hope to expand this project further with other datatypes and adapt the toolbox for different needs and purposes.

ACKNOWLEDGEMENTS

This research has been financed through the PhD grant SFRH/BD/72601/2010 from the Foundation for Science and Technology Portugal for which I am very grateful. I wish to thank my supervisor António de Sousa dias for his guidance in developing DataScapR. Furthermore I would like to thank Dr Robin Laney at the Open University for his revision of this paper.

BIOGRAPHY

Samuel Van Ransbeeck is a composer and is interested in working with extra-musical elements. As such, sonification is a logical road to pursue. In the age of Big Data, bringing Big Data and music together is an exciting endeavour. Samuel recently finished his PhD in Computer Music at the Catholic University of Porto, Portugal where he developed the DataScapR toolbox. Currently, he is working at the Open University in Milton Keynes, UK, where he is expanding DatascapR to use urban data as part of the MK:Smart project.

NOTES

1 Kirilenko, et al. "The Flash Crash"

2 Wagner. "High Frequency Trading"

3 Hamman. "The Technical as Aesthetic"

4 One can consider voyage as the position taking posited by Bourdieu or Jacob's search space. Everything exists, we just search for what we want, see Bourdieu (1993) and Jacob (1995)

5 Hackworth, "John Maeda: Painting by Pixel"

6 Giorgio Sancristoforo, "Gleetchlab Manual"

7 If the data requested is beyond the range of historical prices available through Yahoo Finance, all available data within the range is displayed. Historical prices typically do not go back further than 1970.

8 The spread is calculated by subtracting the open from the close and the high from the low. The resulting values are made absolute, meaning no negative values will exist.

9 The default value is 37, as a reference to Warren Burt's sonification piece *Playing the lottery in plano* (2012)

10 The VST standard, as set out by Steinberg, means "All parameters - the user parameters, acting directly or indirectly on that data, as automated by the host, are 32 bit floating-point numbers. They must always range from 0.0 to 1.0 inclusive, regardless of their internal or external representation." Hence, all mapping modules (except those for the MIDI-events) map between 0 and 1. (Steinberg Media Technologies GmbH 2003)

11 Bach uses Lisp-like linked lists. This implies the use of parentheses to make the hierarchy of the lists clear. Due to space constraints in this paper, I refer to the bach tutorials for an in-depth explication of LLLL's.

12 While allowing smaller divisions (up to 1/100 of a tone), Bach does not support those smaller divisions graphically. Playback however will adhere to the chosen tone division.

13 In the earlier *StockWatch* software I used the standard *StockWatch* object, which only gave access to 5 stock properties. Expanding the properties list was an important motivation in the creation of DataScapR.

14 Yahoo delays the data for some markets for commercial reasons. A list of current delay times can be found on http://finance.yahoo.com/exchanges. To get real-time data, the user has to subscribe to paid services. Hence, in order to keep DataScapR a free toolbox, it was impossible to use non-delayed data. An alternative to Yahoo Finance was Google Finance (started in 2006). However, Yahoo Finance seemed to have more users and getting information was easier, hence I chose to use Yahoo Finance; The API is not officially supported by Yahoo.

BIBLIOGRAPHY

Agrawal, Anup, and Kishore Tandon. 1994. "Anomalies or Illusions? Evidence From Stock Markets in Eighteen Countries." Journal of International Money and Finance 13 (1). Elsevier: 83–106.

Bourdieu, Pierre. 1993. The Field of Cultural Production. Columbia University Press.

Burt, Warren. Letter. 2012. "Questions on Playing Lottery in Plano," December 2.

Goetzmann, William N. 2001. "Patterns in Three Centuries of Stock Market Prices." The Journal of Business 66 (2). University of Chicago Press: 249. doi:10.1086/296603.

Hackworth, Nick. "John Maeda: Painting by Pixel." *Dazed* March, 2006. http://www.dazeddigital.com/artsandculture/article/20241/1/john-maeda-painting-by-pixel

Hamman, Michael. "The Technical as Aesthetic: Technology, Art-making, Interpretation" *Musiques, arts, technologies--Pour une approche critique*, ed. M. Solomos. 2002, archived on michaelhamman.com

Jacob, Bruce L. 1995. "Composing with Genetic Algorithms." In. Banff, Alberta: unknown.

Kirilenko, Andrei A, Albert S Kyle, Mehrdad Samadi, and Tugkan Tuzun. 2011. "The Flash Crash: the Impact of High Frequency Trading on an Electronic Market." SSRN Electronic Journal. doi:10.2139/ssrn.1686004.

Steinberg Media Technologies GmbH. 2003. "What Is a VST Plug-in?" Gersic.com. http://www.gersic.com/vstsdk/.

United States Congress. 1999. Gramm-Leach-Bliley Act. U.S. Government Printing Office. 106.

Van Ransbeeck, Samuel, and Carlos Guedes. 2010. "*Stockwatch*:a Tool for Composition with Complex Data." Parsons Journal for Information Mapping, August, 1–3.

Van Ransbeeck, Samuel, and Carlos Guedes. 2009. "*Stockwatch*: A Tool for Composition with Complex Data." Parsons Journal for Information Mapping 1, no. 3

Wagner, Ryan. 2011. "High Frequency Trading - Financial Ethics - Seven Pillars Institute." Edited by Kara Tan Bhala. Sevenpillarsinstitute.org. Lawrence, KS. February 26. http://sevenpillarsinstitute.org/case-studies/high-frequency-trading.

Weiser, Stanley, and Oliver Stone. 1987. Wall Street. Edited by Oliver Stone. Vol. 126. 20th Century Fox.

Worrall, David. 2009. "The Use of Sonic Articulation in Identifying Correlation in Capital Market Trading Data" Presented at the 15th International Conference on Auditory Display (ICAD2009), Copenhagen, Denmark, May 18-22, 2009

"About Historical Prices." Help.Yahoo.com/Kb/Finance/Historical-Prices-Sln2311.Html. Accessed January 15. https://help.yahoo.com/kb/finance/historical-prices-sln2311.html.